Identifying root causes of web performance degradation using changepoint analysis

Cito, Jürgen ; Suljoti, Dritan ; Leitner, Philipp ; Dustdar, Schahram

Abstract: The large scale of the Internet has offered unique economic opportunities, that in turn introduce overwhelming challenges for development and operations to provide reliable and fast services in order to meet the high demands on the performance of online services. In this paper, we investigate how performance engineers can identify three different classes of externally-visible performance problems (global delays, partial delays, periodic delays) from concrete traces. We develop a simulation model based on a taxonomy of root causes in server performance degradation. Within an experimental setup, we obtain results through synthetic monitoring of a target Web service, and observe changes in Web performance over time through exploratory visual analysis and changepoint detection. Finally, we interpret our findings and discuss various challenges and pitfalls.

Identifying Root Causes of Web Performance Degradation using Changepoint Analysis

Jürgen Cito[1], Dritan Suljoti[2], Philipp Leitner[1], and Schahram Dustdar[3]

[1] s.e.a.l. – software evolution & architecture lab, University of Zurich, Switzerland
{cito, leitner}@ifi.uzh.ch
[2] Catchpoint Systems, Inc., New York, USA
drit@catchpoint.com
[3] Distributed Systems Group, Vienna University of Technology, Austria
dustdar@dsg.tuwien.ac.at

Abstract. The large scale of the Internet has offered unique economic opportunities, that in turn introduce overwhelming challenges for development and operations to provide reliable and fast services in order to meet the high demands on the performance of online services. In this paper, we investigate how performance engineers can identify three different classes of externally-visible performance problems (global delays, partial delays, periodic delays) from concrete traces. We develop a simulation model based on a taxonomy of root causes in server performance degradation. Within an experimental setup, we obtain results through synthetic monitoring of a target Web service, and observe changes in Web performance over time through exploratory visual analysis and changepoint detection. Finally, we interpret our findings and discuss various challenges and pitfalls.

1 Introduction

The large scale of the Internet has offered unique economic opportunities by enabling the ability to reach a tremendous, global user base for businesses and individuals alike. The great success and opportunities also open up overwhelming challenges due to the drastic growth and increasing complexity of the Internet in the last decade. The main challenge for development and operations is to provide reliable and fast service, despite of fast growth in both traffic and frequency of requests. When it comes to speed, Internet users have high demands on the performance of online services. Research has shown that nowadays 47% of online consumers expect load times of *two seconds or less* [7, 14, 21]. With the growth of the Internet and its user base, the underlying infrastructure has drastically transformed from single server systems to heterogeneous, distributed systems. Thus, the end performance depends on diverse factors in different levels of server systems, networks and infrastructure, which makes providing a satisfying end-user experience and QoS (Quality of Service) a challenge for large scale Internet applications. Generally, providing a consistent QoS requires continually collecting data on Web performance on the Web service provider side, in order to observe and track changes in desired metrics, e.g., service response time. Reasons to observe these changes are different in their nature, and range from

detecting anomalies, identifying patterns, ensuring service reliability, measuring performance changes after new software releases, or discovering performance degradation.

Early detection and resolution of root causes of performance degradations can be achieved through monitoring of various components of the system. Monitoring can be classified as either *active* or *passive* monitoring, and, orthogonally, as *external* or *internal*. In active monitoring, monitoring agents are actively trying to connect to the target system in order to collect performance data, whether the system is accessed by real end-users or not. Passive monitoring, on the other hand, only collects measurements if the system is actively used. Internal and external monitoring differentiate in whether the measurements are obtained in systems within the organization's data center or through end-to-end monitoring over the network outside the data center. This has ramifications in terms of what the level of detail of monitoring data that is available. In our experiments, we make use of active, external monitoring, which provides a way of capturing an end user perspective and enables the detection of issues before they affect real users [17].

Whatever the reason to observe changes may be, the measurements are only useful when we know how to properly analyze them and turn our data into informed decisions. The main contribution of this paper is a model for understanding performance data via analyzing how common underlying root causes of Web performance issues manifest themselves in data gathered through *external, active* monitoring. We introduce a taxonomy of root causes in server performance degradations, which serves as the basis for our experiments. Furthermore, we describe the methods and steps we take to obtain our results and explain how the results will be examined and discussed. Following this, we will outline the design of the simulations that will be conducted, as well as the physical experimental setup enabling the simulations. We conclude by providing interpretation of the simulation based results, explaining how we can derive certain conclusions based on exploratory visual analysis and statistical changepoint analysis.

2 Root Causes of Server Performance Degradation

In general, if we consider performance and computation power of a system, we must consider resources that enable computation. These are usually hardware resources, such as processors, memory, disk I/O, and network bandwidth. We also need to consider the ways and methods these resources are allocated and utilized. The demand on resources of a computer system increases as the workload for the application of interest increases. When the demand of the application is greater than the resources that can be supplied by the underlying system, the system has hit its resource constraints. This means the maximum workload of the application has been reached and, typically, the time taken for each request to the application will increase. In case of extreme oversaturation, the system stops reacting entirely. For Web applications and Web services, this translates into poor response times or (temporary) unavailability. A delay in performance as observed through active monitoring can be defined as a negative change in response time at a certain point in time t. This means we look at two observations

of response times x_t and x_{t+1} where $\lambda = |x_t - x_{t+1}| > c$ and c denotes a certain threshold accounting for possible volatility. In the following, this simplified notion of performance delays λ over a threshold c will be used in the description of the elements of the taxonomy.

The underlying causes of performance degradations in Web application and Web service backends are diverse. They differ significantly in the way they manifest themselves in performance data gathered through active monitoring. We propose a simple taxonomy of root causes in Web performance degradation. First, we divide a possible root cause in three main categories, which can be determined through external monitoring: *global delay*, *partial delay*, and *periodic delay*. Further classifications and proper assignment to the main categories in the taxonomy have been derived together with domain experts in the area of Web performance monitoring and optimization. In the following, we provide a brief explanation of the general causes of performance delays in computer systems. We then classify the main categories of our taxonomy by grouping the root causes by the distinguishable effect they have. The taxonomy is depicted in Figure 1, and explained in more detail the following. Note that this taxonomy does by no means raise the claim of completeness. It is rather an attempt to give an overview of common pitfalls that cause slowness in performance.

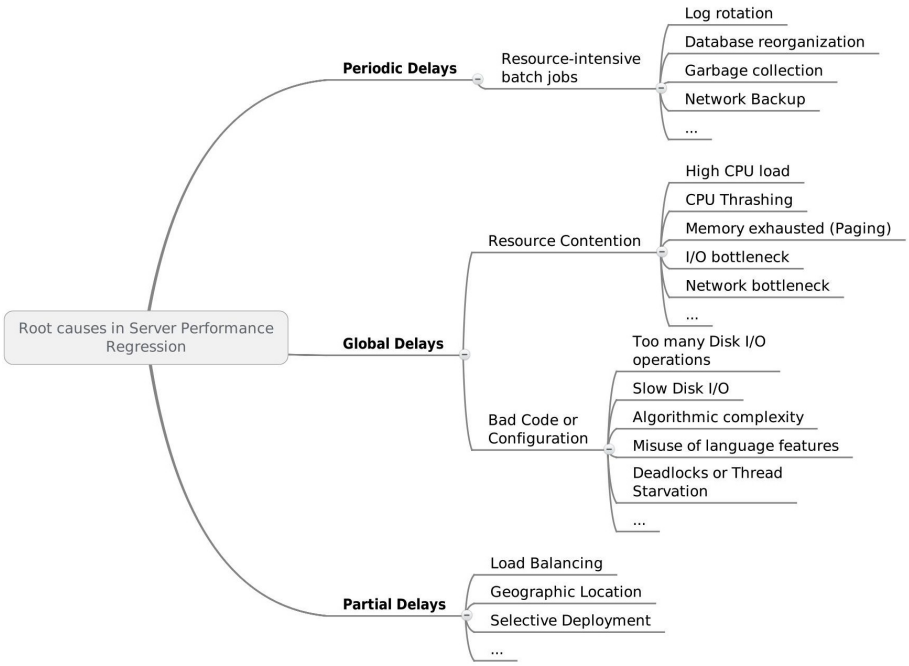


Fig. 1: Taxonomy of Root Causes in Server Performance Degradation

2.1 Global Delay

A global delay means that a change in a new deployment or release of the backend system introduced a significant difference in response time λ, which is higher than a defined threshold c on *all* incoming requests. We distinguish between global delays that are caused through resource contention and code or configuration issues. Global delays caused by resource contention include, for instance, delays due to a bottleneck in disk I/O. In this case, the flow of data from the disk to the application (e.g., via a database query) is contended. This means the query takes longer to perform and return data which causes an overall performance delay. Global delays caused by problems in the application code can for instance be caused by the induction of logical units or algorithms with high computational complexity in the backend service. Alternatively, such global delays may be caused by overzealous synchronization in the application code, which may even lead to temporary service outages when a deadlock situation cannot be immediately resolved by the underlying operating system or virtual machine.

2.2 Periodic Delay

Periodic delays are global delays that are not continuous, but happen, e.g., a few times a day. A periodic delay is mostly not induced through a new deployment or release, but rather through a background process causing the system to use an increased amount of resources. One practical example of such a background job is log rotation. Log rotation is an automated process in server systems administration, where log files that exhibit certain characteristics are archived. The process is usually configured to run as a periodic cronjob to be fully automated. Log rotating often includes archiving and transferring large text files, and, hence, can oversaturate the backend system for a short period of time.

2.3 Partial Delay

Partial delays are global delays that occur only for a subset of all requests, e.g., for a subset of all customers of the Web service. This situation can occur if the application employs a form of redundancy, e.g., load balancing or content distribution networks. In such scenarios, any problems that lead to global delays can potentially be inflicting only one or a subset of all backend servers, hence delaying only those requests that happen to be handled by one of the inflicted backends. Partial delays are interesting, as they are hard to detect (especially if the number of inflicted backends is small).

3 Identifying Root Causes of Performance Degradation

After introducing externally visible classes of root causes of performance degradation (global, partial, periodic), we want to identify characteristics in performance data associated with each class. Furthermore, we want to present statistical methods that are well suited for identifying such changes. Our approach is to generate realistic performance traces for each class through a testbed Web

service, which we have modified in order to be able to inject specific *performance degradation scenarios* associated with each class. We collect data through active monitoring as described in Section 3.1. The specific scenarios we used are described and defined in Section 3.2. Afterwards, we apply the methods described in Section 4.1 to the traces we generated, in order to be able to make general statements about how these methods are able to detect root causes of performance degradation.

3.1 Simulation Design

We consider a simulation model of a simple Web service environment for our experiments. The model consists of the following components: *Synthetic Agent Nodes*, *Scenario Generation Component*, and *Dynamic Web Service Component*. Synthetic agents send out HTTP GET requests every n minutes from m agents and collect response times. In the simulation, we sample every 1 minute resulting in 1 new observation of our system every minute. Each observation is stored in a database with the corresponding timestamp.

The simulation design and its communication channels are depicted in Figure 2. We consider a system with this architecture where requests incoming from the synthetic nodes are governed by a stochastic process $\{Y(t), t \in T\}$, with T being an index set representing time.

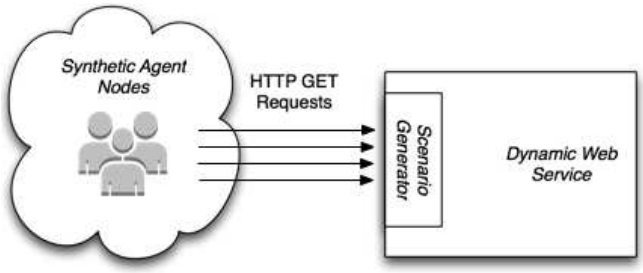


Fig. 2: Simulation Design

Synthetic Agent Nodes We gather data from the *Dynamic Web Service Component* via active, periodic monitoring through *Synthetic Agent Nodes*. A synthetic monitoring agent acts as a client in order to measure availability and performance metrics, such as response time. Every synthetic agent is able to perform active measurements or synthetic tests. An active measurement is a request to a target URL, where subsequently all performance metrics that are available through the response are obtained. When configuring a set of synthetic tests, we can configure the following parameters: (1) URL of the Web service that should be tested; (2) sampling interval, e.g., every n minutes; (3) test duration, i.e., how many sample requests to issue in each run of the test agent.

Scenario Generation Component As only the main classifications of root causes can be identified through synthetic monitoring, changes following the primary notions of *global delays*, *partial delays*, and *periodic delays* (see Section 2) are injected in the simulation. We achieve this by introducing a *Scenario Generation Component* into our model. It functions as an intermediary between the request sent by the synthetic agent nodes and the Web service. Instead of manually injecting faults into our test Web server, we define a set of scenarios that, subsequently, reflect the desired scenario, i.e., the faults over time in our system. The scenarios need to reflect performance degradation and performance volatility within a certain system, i.e., a single Web server. The Scenario Generation Component also needs to take into account possible geographic distribution of the agents, as well as load balancing mechanisms. In the following, we introduce a formal model for defining scenarios that reflects these notions that can be used to formally describe scenarios.

We consider a given set of parameters to compose a complete scenario within our simulation model. Within a scenario, we need to be able to specify how performance metrics (i.e., response times) develop over time, as well as synthetic agents that are actively probing our target system.

- A development $D \in \mathcal{D}$ maps from a certain point in $t \in T$ of the stochastic process $\{Y(t), t \in T\}$ (driving the requests of the synthetic agent nodes) to an independent random variable $X_i \in \mathcal{X}$, where $\mathcal{X}$ being the set of possible random variables (Equation 1).

$$D : T \mapsto \mathcal{X} \tag{1}$$

  where $X_i \in \mathcal{X}$ and $\forall\ X_i \sim \mathrm{U}(a, b)$
- On the top-level, we define a scenario $\mathcal{S}$ that describes how the target system that is observed by each synthetic agent $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ develops over time. Each agent observes a development in performance $D_i \in \mathcal{D}$, with $\mathcal{D}$ being the set of all possible developments (Equation 2).

$$\mathcal{S} : \mathcal{A} \mapsto \mathcal{D} \tag{2}$$

This formalization allows us to express any performance changes (either positive or negative) as a classification of performance developments over time attributed to specific synthetic agents. More accurately, it models a performance metric as a uniformly distributed random variable of a system at any given time point $t \in T$. Specifying an assignment for every point in time is a tedious and unnecessary exercise. In order to define scenarios in a more efficient and convenient way, we introduce the following notation for developments $D \in \mathcal{D}$:

- Simple Developments: $[X_0, t_1, X_1, ..., t_n, X_n]$ defines a *simple development* as a sequence of independent random variables X_i and points in time t_i, further defined in Equation 3.

$$[X_0, t_1, X_1, ..., t_n, X_n](t) = \begin{cases} X_0 & 0 \leq t < t_1 \\ X_1 & t_1 \leq t < t_2 \\ \vdots & \\ X_n & t_n \leq t \end{cases} \tag{3}$$

This allows us to easily define developments X_i in terms of time spans $t_{i+1} - t_i \geq 0$ within the total time of observation. The last development defined through X_n remains until the observation of the system terminates.

– Periodic Developments: A *periodic development* is essentially a simple development, which occurs in a periodic interval p. It is preceded by a "normal phase" up until time point n. The "periodic phase" lasts for $p-n$ time units until the development returns to the "normal phase". A periodic development $[X_0, n, X_1, p]^*$ is defined in Equation 4.

$$[X_0, n, X_1, p]^*(t) = \begin{cases} X_1 & \text{for } kp + n \leq t < (k+1)p \\ X_0 & \text{otherwise} \end{cases} \tag{4}$$

where $k \geq 0$.

Figure 3 depicts how a periodic development can be seen over time with given parameters X_0 as the "normal phase" random variable, X_1 as the "periodic phase" random variable, n to define the time span for a "normal phase", p as the periodic interval and $(p-n)$ as the time span for the "periodic phase".

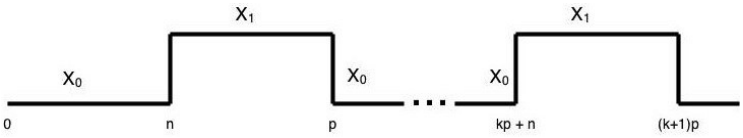


Fig. 3: Depiction of the periodic development scheme defined in Equation 4

These two defined functions allow us to conveniently define scenarios which adhere to our notions of global, partial and periodic delays. We can now define changes in performance at certain points in time in a declarative way, as well as define changes in periodic time intervals that result in changes in response times for a specific amount of time.

Dynamic Web Service Component The *Dynamic Web Service Component* works together with the *Scenario Generation Component* to achieve the reflection of performance issues for the *Synthetic Agent Nodes*. In order to do so, it offers an endpoint that simulates delays over parameters passed from a specific scenario script. This means that the declared scenarios are executed within the Web service component and thus simulate workload through the parameters given in the scenarios.

3.2 Simulation Scenarios

Here, we formally define the parameters that actually form a certain scenario, and give an example of a real-life situation that would lead to such a performance behavior. The aim of each scenario is to properly represent one of global, partial or periodic delay.

Global Delay A global delay is the introduction of a significant difference in response time on all incoming requests, i.e., it affects all users accessing the resource in question.

Example Scenario Use Case A new feature needs to be implemented for a new release. A junior developer in charge of the new feature introduces a new (slow) database query, causing significantly higher overall response times. The slow query is not caught in QA (Quality Assurance) and the new release is deployed to all users.

Scenario Parameters The parameter for this delay is given in Equation 5. Further, for every index i, we define the initial response time range as in Equation 6, as well as the range for the global change over all agents in Equation 7.

$$\mathcal{S}_G = \{a_i \mapsto [X_{a_i,0}, 420, X_{a,1}] \mid a_i \in \{a_1, a_2, a_3, a_4, a_5\}\} \tag{5}$$

$$\begin{aligned} X_{a_1,0} \sim \mathrm{U}(90, 115), X_{a_2,0} \sim \mathrm{U}(100, 130), X_{a_3,0} \sim \mathrm{U}(110, 140), \\ X_{a_4,0} \sim \mathrm{U}(95, 110), X_{a_5,0} \sim \mathrm{U}(100, 110) \end{aligned} \tag{6}$$

$$X_{a,1} \sim \mathrm{U}(150, 175) \tag{7}$$

Partial Delay A partial delay scenario consists of requests that, at some point in time, cause a delay on a subset of the incoming requests.

Example Scenario Use Case A Web application sits behind a load balancer handling 5 servers. One of the servers encounters unexpected hardware issues, which result in higher response times. The balancer uses "Round Robin" as its load balancing algorithm [22]. 20% of all users perceive the application with higher response times.

Scenario Parameters The parameter for this delay is defined in Equation 8. For every index i we define the initial response time range as in Equation 9, as well as the range for the partial change for agent a_5 in Equation 10.

$$\mathcal{S}_P = \{a_i \mapsto [X_{a_i,0}, \infty] | a_i \in \{a_1, a_2, a_3, a_4, a_5\}, a_5 \mapsto [X_{a_5,0}, 360, X_{a_5,1}]\} \tag{8}$$

$$\begin{aligned} X_{a_1,0} \sim \mathrm{U}(115, 125), X_{a_2,0} \sim \mathrm{U}(115, 120), X_{a_3,0} \sim \mathrm{U}(120, 145), \\ X_{a_4,0} \sim \mathrm{U}(105, 115), X_{a_5,0} \sim \mathrm{U}(110, 120) \end{aligned} \tag{9}$$

$$X_{a_5,1} \sim \mathrm{U}(140, 165) \tag{10}$$

Periodic Delay A periodic delay takes place when, due to a (background) process, resource contention occurs and, subsequently, higher usage of hardware resources leads to higher response times for a certain amount of time. This scenario addresses those processes that are (usually) planned ahead and are executed within a specific interval.

Example Scenario Use Case Log files of an application make up a large amount of the server's disk space. The system administrator creates a cron job to process older log files and move them over the network. The process induces heavy load on CPU (processing) and I/O (moving over network), which result into temporarily higher response times. The cron job is configured to take place in periodic intervals to ensure the server has enough disk space.

Scenario Parameters The parameter for this periodic delay is defined in Equation 11. The response time ranges are defined as in Equation 12.

$$\mathcal{S}_{PD} = \{a_1 \mapsto [X_0, 45, X_1, 65]^*\} \tag{11}$$

$$X_0 \sim \mathrm{U}(95, 115), X_1 \sim \mathrm{U}(160, 175) \tag{12}$$

4 Experiments

We now describe the methods of analysis and execution of the experiments introduced in Section 3, as well as the concrete results we achieved.

4.1 Methods of Analysis

The specified scenarios are executed within a physical testbed (described in Section 4.2) and results are analyzed and interpreted. The following sections outline the methods of analysis that are applied to the results.

Exploratory Data Analysis Our first analysis approach is to examine the time series of monitored response times over time visually over graphs in order to explore and gain further understanding on the effect specific underlying causes have on the resulting data. We also determine what kind of statistical attributes are well suited to identify key characteristics of the data sample and for humans to properly observe the performance change. For this initial analysis we plot the raw time series data[4] as line charts. This allows for visual exploratory examination, which we further complement with proper interpretations of the data displayed in the visualization that correlates with induced changes in the server backend.

[4] Raw in this context means that we will not smooth the data by any means and will not apply statistical models in any way.

Statistical Analysis After manually inspecting the time series over visual charts, we evaluate the observation of performance changes by the means of statistical changepoint analysis. Specifically, we evaluate algorithms that are employed in the R [20] package "changepoint". This approach makes sense, as we are not interested in the detection of spikes or other phenomena that can be considered as outliers in the statistical sense, but rather in the detection of fundamental shifts in our data that reflect a longer standing performance degradation. We also want to keep false positives low, and determine whether a change is actually a change that implies a root cause that requires some kind of action. Thus, we also need to determine the magnitude of the change. For this, we recall the simple view on delays we introduced in our taxonomy: $\lambda = |x_t - x_{t+1}| > c$. We adapt this model of change as follows. Instead of comparing two consecutive data points x_t and x_{t+1}, we compare the changes in the mean at the time point where changepoint have been detected by the algorithm. In other words, we compute the difference between the mean of the distribution before the detected changepoint occurred, $\mu_{<\tau}$, and the mean of the distribution after the detected changepoint occured, $\mu_{>\tau}$, where τ denotes the changepoint. This difference is then denoted as λ, and can be defined as $\lambda = |\mu_{<\tau} - \mu_{>\tau}| > c$ via replacing two variables.

The threshold c is challenging to determine optimally. When c is set up too high, legitimate changes in performance that were caused by problems may not be detected and the system is in risk of performance degradation. When c is defined unnecessarily sensitive, the monitoring system is prone to false positives. The value of the threshold depends on the application and must be either set by a domain expert or be determined by statistical learning methods through analysis of past data and patterns. Sometimes it is useful to compare new metrics in relation to old metrics, this is a very simple way of statistical learning through past data. In the conducted experiments, we set the threshold as $c = \mu_{<\tau} \cdot 0.4$. This means that if the new metric after the changepoint $\mu_{>\tau}$ is 40% above or below the old metric $\mu_{<\tau}$, the change is considered a *real change* as opposed to a *false positive*. If we want to consider positive changes as well, the calculation of the threshold must be extended in a minor way to not yield into false negatives due to a baseline that is too high: $c = \min(\mu_{>\tau}, \mu_{<\tau}) \cdot 0.4$.

Note that, in the case of this paper, the threshold 40% of the mean was chosen after discussions with a domain expert, as it is seen as an empirically estimated baseline. In practice, a proper threshold depends on the type of application, SLAs, and other factors and is usually determined by own empirical studies.

4.2 Testbed Setup

The experiments based on the described simulation model were executed in a local testbed consisting of 5 *Synthetic Agent Nodes* and 1 *Dynamic Web Service Component*. The agents operate in the local network as well. Each of the 5 nodes sends out a request every 5 minutes that is handled by a scheduler that uniformly distributes the requests over time. This results into 1 request/minute that is being send out by an agent that records the data. The physical node setup consists of Intel Pentium 4, 3.4 GHz x 2 (Dual Core), 1.5 GB RAM on Windows and is running a Catchpoint agent node instance for monitoring. The

Web service running on the target server has been implemented in the Ruby programming language and runs over the HTTP server and reverse proxy nginx and Unicorn.The physical web server setup is the same as the synthetic agent node setup, but is running on Ubuntu 12.04 (64 bit). During simulation, a random number generator is applied to generate artificial user behavior as specified in the distributions, which can be represented efficiently with common random numbers [12]. However, as with deterministic load tests, replaying user behavior data may not always result into the same server response. Even with the server state being the same, server actions may behave nondeterministically. To adhere the production of independent random variables that are uniformly distributed we use MT19937 (Mersenne twister) [18] as a random number generator.

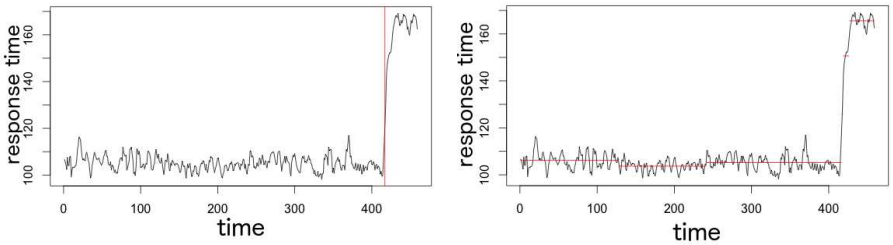
4.3 Results and Interpretation

We now discuss the results of our scenario data generation, and the results of applying the statistical methods described in Section 4.1. At first, we display a raw plot of the resulting time series data without any filters and interpret its meaning. Further, we apply a moving average smoothing filter (with a window size $w = 5$) to each resulting time series and conduct a changepoint analysis.

Global Delay The global delay forms the basis of our assumptions on how performance changes can be perceived on server backends. Both, the partial and periodic delay, are essentially variations in the variables time, interval and location of a global delay. Hence, the findings and interpretations of this section on global delays are the foundation for every further analysis and discussion.

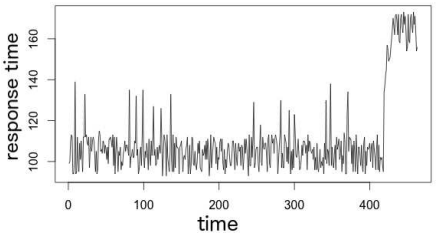
Exploratory Visual Analysis In Figure 4c, we see the results for the global delay scenario. We can clearly see the fundamental change in performance right after around 400 minutes of testing. The data before this significant change does seem volatile. There are a large amount of spikes occurring, though most of them seem to be outliers that might be caused in the network layer. None of the spikes sustain for a longer period of time, in fact between the interval around 150 and 250 there seem to be no heavy spikes at all. The mean seems stable around 115ms in response time and there is no steady increase over time that might suggest higher load variations. Thus, we can conclude that the significant performance change has occurred due to a new global release deployment of the application.

Statistical Changepoint Analysis We apply changepoint analysis to the smoothed time series with a moving average window size of 5. In Figure 4a, we can immediately see how the smoothing affected the chart, compared to the chart with the raw data in Figure 4c: The spikes, i.e., the random noise, have been canceled out to a certain extent, making the main signal stronger and easier to identify. This makes it easier for our statistical analysis to focus on our signal and detect the proper underlying distributions. While this is definitely a pleasant effect of every smoothing technique, we also need to keep in mind that every model that we apply contains its own assumptions and own errors that need to be considered. What can further be seen in Figure 4a is the changepoint in the variance,



(a) Variance Changepoint

(b) Mean Changepoint



(c) Global delay results raw time series

Fig. 4: Global Delay results

denoted by a vertical red line at the changepoint location. Table 1 contains the numerical results of the changepoint in variance analysis and indicates the estimation for the changepoint τ_1 at 422. This number coincides approximately with our own estimation we concluded in the previous section.
Next, we look at Figure 4b, where the change in the mean is indicated by horizontal lines depicting the mean value for each segment that has been detected. This method has detected more changepoints than the previous analysis, which can be not clearly seen in Figure 4b due to the very small change in the mean. The estimated numerical values for the changepoints are listed in Table 2. The table also lists the mean values, as well as the calculated threshold $c = \mu_{<\tau} \cdot 0.4$. The last column also states whether or not a detected changepoint in the mean is an *actual changepoint* (CP) as defined by the notion of *significant change* where $\lambda = |\mu_{<\tau} - \mu_{>\tau}| > c$, or a *false positive* (FP). Only one of the estimated changepoints has been identified as CP when considering the threshold c. This shows that detecting a fundamental change is a difficult undertaking, especially considering non-parametric statistical analysis, as in our case. Post-processing and analysis of the estimated changepoints and its according mean values is important to avoid false positives.

Partial Delay Partial delays are global delays that only occur on a certain subset of requests and, therefore, need different techniques to properly examine the time series data and diagnose a performance degradation. Experiments on simulating partial delays have found that detection of a changepoint in partial delays, or even the visual detection of performance change, is not at all trivial.

Exploratory Visual Analysis As before, we plot the time series data and look for changes in our performance. In Figures 5a and 5b, we see rather stable time

τ	$\sigma^2_{<\tau}$	$\sigma^2_{>\tau}$
422	10.695	67.731

Table 1: Variance CP for $\mathcal{S}_G$

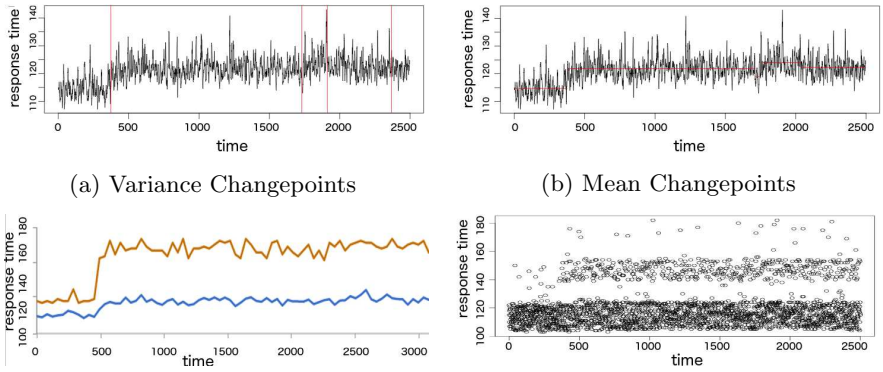
τ	$\mu_{<\tau}$	$\mu_{>\tau}$	λ	c	CP/FP
127	106.18	103.7	2.48	42.47	FP
241	103.7	105.32	1.62	41.48	FP
366	105.32	110.85	5.53	42.12	FP
374	110.8	103.62	7.23	44.32	FP
421	103.62	150.65	47.03	41.44	**CP**
427	150.65	165.62	14.97	60.62	FP

Table 2: Mean CP for $\mathcal{S}_G$

series charts, relatively volatile (spiky), due to the higher amount of conducted tests and random network noise. Around the time points 460-500 and 1600-1900, we see a slight shift, but nothing alarming that would be considered a significant change. From this first visual analysis, we would probably conclude that the system is running stable enough to not be alerted. However, that aggregation hides a significant performance change. The convenience, or sometimes necessity, of computing summary statistics and grouping data to infer information this time concealed important facts about the underlying system. In order to detect this performance change, we have to look at additional charts and metrics.

In Figure 5c, we plot the same aggregation as in Figures 5a and 5b, but also plot the 90th percentile of the data to come to a more profound conclusion: There actually has been a performance change that is now very clear due to our new plot of the 90th percentile. While this can mean that percentiles also show temporary spikes or noise, it also means that if we see a significant and persisting shift in these percentiles, but not in our average or median, that a subset of our data, i.e., a subset of our users, indeed has experienced issues in performance and we need to act upon this information. Another way of detecting issues of this kind is to plot all data points in a scatterplot. This allows us to have an overview of what is going on and to identify anomalies and patterns more quickly. As we can see in Figure 5d, a majority of data points is still gathered around the lower response time mean. But we can also see clearly that there has been a movement around the 400 time point mark that sustains over the whole course of the observation, building its own anomaly pattern.

Statistical Changepoint Analysis Analyzing both the changepoints in the variance in Table 3 and Figure 5a, as well as the changepoints in the mean in Table 4 and Figure 5b yields no surprise following our initial exploratory visual analysis. The changes that have been identified are not significant enough to be detected through the mean and the variance respectively. Although we need to point out that both analyses actually detected the actual significant changepoint around the time point 374-376, but are disregarded as false positives by our post-processing step of checking the threshold. Thus, our post-process actually resulted into a *false negative*. This is usually a sign that an indicator (in our case the threshold c) needs to be adjusted or rethought completely. However, before this kind of decision can be made, more information has to gathered on how this indicator has performed in general (i.e., more empirical evidence on false negatives, ratio between false positives and false negatives, etc.).
In order for our regular statistical analysis process, as applied previously, to properly work we need a further pre-processing step. Neither mean nor variance



(a) Variance Changepoints

(b) Mean Changepoints

(c) 90th percentile and mean plot

(d) Partial Delay Scatterplot

Fig. 5: Partial Delay results

can detect the performance change, therefore, we need to consider a different metric. In our exploratory analysis, we concluded that the 90th percentile was able to detect the change. Thus, we need to apply our changepoint analysis to percentiles in order to detect a significant shift for partial delays.

τ	$\sigma^2_{<\tau}$	$\sigma^2_{>\tau}$
374	12.88	24.25
1731	24.25	12.43
1911	12.43	6.63

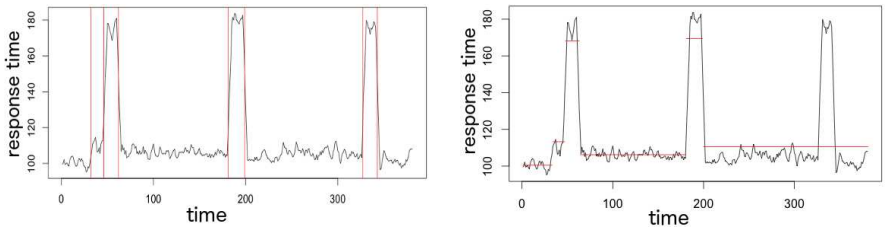
Table 3: Variance CP for $\mathcal{S}_P$

τ	$\mu_{<\tau}$	$\mu_{>\tau}$	λ	c	CP/FP
376	114.74	121.92	7.18	45.88	FP
1710	121.92	118.91	3.01	48.76	FP
1756	118.91	124.1	5.19	47.56	FP
2035	124.1	122.27	1.83	49.64	FP

Table 4: Mean CP for $\mathcal{S}_P$

Periodic Delay Periodic delays are either global or partial delays that occur in specific intervals, persist for a certain amount of time, and then performance goes back to its initial state.

Exploratory Visual Analysis For this experiment we recorded enough values to result into three periods that indicate a performance degradation for a certain amount of time before returning back the system returns to its normal operation. The intuition we have on periodic delays can be clearly observed in Figure 6. Between the phases, we have usual performance operation with usual volatility, and in between we see fundamental shifts that persist for approximately 20 minutes before another shift occurs. Seeing periodic delays is fairly simple, as long as we are looking at a large enough scale. If we would have observed the time series within the periodic phase, before the second shift to the normal state occurred, we might have concluded it to be a simple global delay. Thus, looking at time series at a larger scale might help to identify periodic delays that would have otherwise been disregarded as short-term trends or spikes.



(a) Variance Changepoint (b) Mean Changepoint

Fig. 6: Periodic Delay results

Statistical Changepoint Analysis While the exploratory visual analysis in periodic delays was straight forward, the changepoint analysis brings some interesting insights. Figure 6a and Table 5 show the analysis for the changepoints in the variance, which mostly coincide with our visual assessment. Merely the first detected changepoint in the variance is a false negative.
The changepoint in the mean yields more interesting results, as seen in Table 6 Contrary to the results in the other experiments the number of false positives is very low at only one. When looking at the result in Figure 6b, we can observe another interesting phenomena we have not seen before, a false negative. The third period was not detected as a changepoint in the mean by the algorithm, although it was detected by the changepoint in the variance method. This means, in order to avoid false negatives, we should apply both analyses for changepoint in the mean and variance.

τ	$\sigma^2_{<\tau}$	$\sigma^2_{>\tau}$
32	5.21	20.22
46	20.22	156.25
62	156.25	11.73
181	11.73	233.96
199	233.96	16.3
327	16.3	180.3
343	180.3	25.16

Table 5: Variance CP for $\mathcal{S}_{PD}$

τ	$\mu_{<\tau}$	$\mu_{>\tau}$	λ	c	CP/FP
33	100.47	113.27	12.8	40.18	FP
47	113.27	168.11	54.84	45.31	**CP**
63	168.11	106.18	61.93	42.47	**CP**
180	106.18	169.35	63.17	42.46	**CP**
199	169.35	110.52	58.83	44.20	**CP**

Table 6: Mean CP for $\mathcal{S}_{PD}$

4.4 Threats to Validity

In this research, we employ a simplified simulation model in order to show how significant performance changes can be detected in continuously observed data of a constructed target system. The creation and design of a simulation model comes with inherent risk concerning the validity of the results when applied to real-life systems. Most importantly, in our simulation design, we do not actively inject the notion of web traffic workloads (as has been, for instance, taken in consideration in [16]), but rather simulate the variability of workloads (and

therefore response times) in web services through specific scenarios parameters. Further, the notions of network traffic and network volatility in wide area networks (WAN) are completely omitted to limit the interference of network noise in the recorded data. Of course, noise is still present even in LANs, but is limited to very few factors within the data center, as opposed to the possible factors in WANs. This also means that there is, for instance, no increased response time due to DNS resolution in our experiments.

5 Related Work

Analyzing performance data observed through continuous monitoring in distributed systems and web services has produced a large body of research dealing with statistical modeling of underlying systems, anomaly detection and other traditional methods in statistics to address problems in performance monitoring and analysis. For instance, Pinpoint [8] collects end-to-end traces of requests in large, dynamic systems and performs statistical analysis over a large number of requests to identify components that are likely to fail within the system. It uses a hierarchical statistical clustering method, using the arithmetic mean to calculate the difference between components by employing the Jaccard similarity coefficient [13]. [11] makes use of statistical modeling to derive signatures of systems state in order to enable identification and quantification of recurrent performance problems through automated clustering and similarity-based comparisons of the signatures. [1] proposes an approach which identifies root causes of latency in communication paths for distributed systems using statistical techniques. An analysis framework to observe and differentiate systemic and anomalous variations in response times through time series analysis was developed in [9]. A host of research from Borzemski *et al.* highlights the use of statistical methods in web performance monitoring and analysis [2–6]. The research work spans from statistical approaches to predict Web performance to empirical studies to assess Web quality by employing statistical modeling. [19] suggests an approach to automated detection of performance degradations using control charts. The lower and upper control limits for the control charts are determined through load testing that establish a baseline for performance testing. The baselines are scaled employing a linear regression model to minimize the effect of load differences. [10] proposes an automated anomaly detection and performance modeling approach in large scale enterprise applications. A framework is proposed that integrates a degradation based transaction model reflecting resource consumption and an application performance signature that provides a model of runtime behavior. After software releases, the application signatures are compared in order to detect changes in performance. [17] conducted an experimental study comparing different monitoring tools on their ability to detect performance anomalies through correlation analysis among application parameters. In the past, we have also applied time series analysis to the prediction of SLA violations in Web service compositions [15].

6 Conclusion and Future Work

A taxonomy of root causes in performance degradations in web systems has been introduced, which was further used to construct scenarios to simulate issues in web performance within an experimental setup. In a series of simulations, we measured how performance metrics develop over time and presented the results. Furthermore, we provided analysis and interpretation of the results. Following the work presented in this paper, there are possible improvements and further work we were not able to address sufficiently. Performance data gathered through external synthetic monitoring only allows for a black box view of the system and is often not sufficient for in-depth root cause analysis of performance issues. Combining data from external monitoring and internal monitoring in order to automate or assist in root cause analysis and correlation of issues is a possible approach that should be considered. The simulation and analysis in this paper is limited to performance issues on the server. Further work might include extending the taxonomy of root causes and simulation scenarios to also represent frontend performance issues.

Acknowledgements

The research leading to these results has received partial funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement 610802 (CloudWave). Numerical simulation was carried out during a research visit of the first author at Catchpoint Systems, Inc., New York, USA. The authors would like to thank Prof. Gernot Salzer for his comments on earlier versions of this work.

References

1. Marcos K Aguilera, Jeffrey C Mogul, Janet L Wiener, Patrick Reynolds, and Athicha Muthitacharoen. Performance debugging for distributed systems of black boxes. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 74–89. ACM, 2003.
2. Leszek Borzemski. The experimental design for data mining to discover web performance issues in a wide area network. *Cybernetics and Systems*, 41(1):31–45, 2010.
3. Leszek Borzemski and Maciej Drwal. Time series forecasting of web performance data monitored by mwing multiagent distributed system. In *ICCCI (1)*, pages 20–29, 2010.
4. Leszek Borzemski and Anna Kaminska-Chuchmala. Knowledge discovery about web performance with geostatistical turning bands method. In *KES (2)*, pages 581–590, 2011.
5. Leszek Borzemski and Anna Kaminska-Chuchmala. Knowledge engineering relating to spatial web performance forecasting with sequential gaussian simulation method. In *KES*, pages 1439–1448, 2012.
6. Leszek Borzemski, Marta Kliber, and Ziemowit Nowak. Using data mining algorithms in web performance prediction. *Cybernetics and Systems*, 40(2):176–187, 2009.

7. Anna Bouch, Allan Kuchinsky, and Nina Bhatti. Quality is in the eye of the beholder: meeting users' requirements for internet quality of service. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 297–304. ACM, 2000.
8. Mike Y Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 595–604. IEEE, 2002.
9. Yingying Chen, Ratul Mahajan, Baskar Sridharan, and Zhi-Li Zhang. A provider-side view of web search response time. *SIGCOMM Comput. Commun. Rev.*, 43(4):243–254, August 2013.
10. Ludmila Cherkasova, Kivanc Ozonat, Ningfang Mi, Julie Symons, and Evgenia Smirni. Automated anomaly detection and performance modeling of enterprise applications. *ACM Transactions on Computer Systems (TOCS)*, 27(3):6, 2009.
11. Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. Capturing, indexing, clustering, and retrieving system history. In *ACM SIGOPS Operating Systems Review*, volume 39, pages 105–118. ACM, 2005.
12. Russell G Heikes, Douglas C Montgomery, and Ronald L Rardin. Using common random numbers in simulation experiments - an approach to statistical analysis. *Simulation*, 27(3):81–85, 1976.
13. Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.
14. Andrew King. *Speed up your site : Web site optimization*. New Riders, Indianapolis, Ind, 2003.
15. Philipp Leitner, Johannes Ferner, Waldemar Hummer, and Schahram Dustdar. Data-Driven and Automated Prediction of Service Level Agreement Violations in Service Compositions. *Distributed and Parallel Databases*, 31(3):447–470, 2013.
16. Zhen Liu, Nicolas Niclausse, Cesar Jalpa-Villanueva, and Sylvain Barbier. Traffic Model and Performance Evaluation of Web Servers. Technical Report RR-3840, INRIA, December 1999.
17. Joao Paulo Magalhaes and Luis Moura Silva. Anomaly detection techniques for web-based applications: An experimental study. In *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, pages 181–190. IEEE, 2012.
18. Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.
19. Thanh HD Nguyen, Bram Adams, Zhen Ming Jiang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. Automated detection of performance regressions using statistical process control techniques. In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*, pages 299–310. ACM, 2012.
20. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.
21. Forrester research. Ecommerce web site performance today: An updated look at consumer reaction to a poor online shopping experience, August 2009.
22. Behrooz A. Shirazi, Krishna M. Kavi, and Ali R. Hurson, editors. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.